tions in the on-chip function table are listed in the same order as in the flash function table. Again, functions may be ordered to maximize locality.

[0087] As represented by block **406**, the program code may be defined and/or modified to support function calls to code stored in either the internal memory or the external memory. This may involve, for example, modifying function calls for functions that are not contained in romOnlyFunctions.txt.

[0088] In some embodiments the compiler initially compiles the program code and determines which functions are placed in the internal memory and, if applicable, external memory. In some embodiments, to enable a function that is stored in internal memory to be replaced with a function stored in external memory, function calls may be expanded to enable a relatively simple code swap. Various code implementations may be used to provide this functionality. In general, the code implementation depends on the specific processor in the system. One example of such an expansion for an Open RISC processor follows:

[0089] Original jump-and-link function call:

[0090] jal F_function

[0091] New load-word and jump-and-link function call:

[0092] lw r**1**, TABLE_OFST(r**28**)

[0093] jalr r**1**

[0094] Here, a register (r**28**) contains the base address of the function table. The base address will either point to the internal memory or to the external memory. The TPM will first load the function pointer from the function table, and will then jump-and-link to the function address in register r**1**.

[0095] In some embodiments there is a unique table offset ("TABLE_OFST") for each function that resides in a function table. For functions that reside in both the internal and external function tables, the TABLE_OFST is the same. Register r**28** (or any other register or memory location may be used for this purpose) may be reserved during compilation. The firmware will initialize this register to either point to the internal or external function table.

[0096] The use of code expansion as described above may have a relatively minor impact on the performance of the system. In general, jump-and-link instructions make up a relatively small amount of the code. In a secure-code implementation, code size may increase by this amount due to the function pointer table access. In addition, there may an increase in latency if the instruction cache is not used to authenticate and decrypt the flash function pointer table and flash rodata access.

[0097] As represented by block **408**, the internal and external function tables may be generated by, for example, calculating function pointers for each function. This procedure is discussed in more detail below.

[0098] The internal function table may be used before secure code has been loaded into flash or when a new secure code image is being loaded. When the internal function table is used, all called functions are contained in on-chip ROM. In some embodiments the internal function table includes boot routines, flash access functions, self-test functions, authorization routines, and field upgrade commands. In some embodiments this function table may only be generated during a full-layer or metal tape-out.

[0099] The external function table may be used once secure code has been loaded into external flash. The TPM uses the external function table to determine whether a given function resides in the on-chip ROM or the external flash memory. Original functions that have been modified may reside in both the on-chip and external flash function tables. In this case, the flash function table may indicate that the newer, modified function stored in external memory should be used. Even if all of the code initially resides in on-chip ROM, the flash function table may be generated so that functions may be replaced in future releases.

[0100] At block **410** the compiler completes the compilation of the program code (including the functions) to generate the final machine code. In the event some functions are to be stored in the external memory, machine code may be generated for both the internal memory and the external memory as discussed above.

[0101] As represented by block **412**, at some point during the manufacturing process a verification key (and optionally an encryption key) may be generated for the code update process. For example, in some embodiments cryptographic processor(s) in a secure environment (e.g., a FIPS level 3 environment such as a hardware security module) generates public and private key pair(s) to be used with all TPM devices that the secure environment supports. The secure environment then provides the public key(s) to the manufacturer so that the manufacturer may install the public keys in each TPM. The private keys, on the other hand, are protected from disclosure. For example, the signing (and encryption) keys may be stored in the hardware security module.

[0102] As represented by block **414**, the manufacturer tapes-out the chip and incorporates the internal secure code information into the TPM device (e.g., chip). The internal secure code information may include, for example, the internal function table, the internal secure code, the internal rodata and the public verification (and optionally decryption) keys. In some embodiments the internal secure code information may be incorporated into the internal ROM during a full tape-out or a metal tape-out of the chip. In other embodiments the code may be loaded into a chip after it is manufactured.

[0103] As represented by block **416**, the manufacturer also generates a flash secure code descriptor. The flash secure code descriptor may be used to describe the code stored in the external flash. For example, the secure code descriptor may contain information that may be required by the TPM to correctly initialize secure code hardware. In addition, the secure code descriptor may be used to hold verification (and optionally decryption) keys.

[0104] The following describes several examples of fields that may be defined for one embodiment of a secure code descriptor. One field may indicate that secure code firmware stored in flash is valid. This field may be securely set in flash by the upgrade complete sub-command as discussed below. This field may be securely cleared in flash by the upgrade start sub-command as discussed below. This field may be set by the TPM.

[0105] Other fields may store, for example, the version of the upgrade information, the most-significant 4 bytes of the